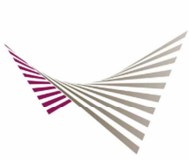
Universitat de Lleida

TREBALL FINAL DE GRAU



ESCOLA
POLITÈCNICA SUPERIOR
UNIVERSITAT DE LLEIDA
INSPIRING THE FUTURE

Estudiant: Alba Lamas Varela

Titulació: Grau en Enginyeria Informàtica

Títol de Treball Final de Grau: **Classificació de malware d'Android utilitzant xarxes neuronals**

Director/a: **Jordi Planes Cid i Daniel Gibert Llauradó**

Presentació

Mes: Juliol

Any: 2018

Resum

En aquest treball es pretén fer un estudi de l'ús de xarxes neuronals per la classificació de programari maliciós del sistema operatiu de smartphones Android.

Per fer-ho, es decideix construir un graf a partir de les crides i el flux de l'aplicació i buscar una forma de representar-lo perquè una xarxa neuronal pugui entrenar amb grafs com entrada.

Entrenar una xarxa neuronal amb grafs resulta ser un segon repte dintre de la implementació de la part pràctica del treball, ja que no es pot fer directament i s'han de reestructurar les dades que forma que no es perdi informació.

Índex

Índex de figures

Agraïments

Començaré els agraïments pel final, agraint a totes aquelles persones que van pensar que no arribaria fins on he arribat, que pensaven que no seria res a la vida i que no podria "fer una O amb un canut". Gràcies a elles he lluitat i lluito per demostrar el contrari, que sí que puc i que s'equivocaven del tot. També em disculpo amb les persones que no menciono, però la vida és així ... Sempre queda algú darrere.

Dono gràcies a un dels meus tutors de la primària, Juan. Recordo que tots els alumnes es queixaven d'ell per ser molt exigent, però a mi em va ensenyar el valor de la feina ben feta. De la mateixa forma, agraeixo a Fernando, que va estar al meu costat en una època de transició que no va ser gens bona i que gràcies al seu suport, comprensió i afecte, vaig poder superar sense por. Dono gràcies a Lola, per ser tan dolenta i baixar-me l'autoestima a cada poc... per no creure en mi, com van fer altres... Gràcies a ella i a altres persones cada dia em supero més i més, per demostrar que, tal com em repetia (i encara em repeteix) la meva mare, arribaré allí on jo vulgui.

A Tati, Miriam i Carlos, per estar amb mi, tot i la distància.

Als meus professors de sisè de primària, el primer curs que vaig fer a Catalunya, per recolzar-me i alimentar les meves ganes de saber-ne més i més. La veritat és que vaig tenir molta sort amb ells, perquè entenien les meves necessitats i m'animaven a continuar endavant.

De l'època d'institut vull agrair a Cristina, una tutora que vaig tenir un

any per entendre'm sempre, a Enrique, el meu professor de matemàtiques de batxillerat al qual sempre estaré agraïda per ser tan bon professor. També dono les gràcies a Aureli, gràcies per cada recomanació que m'has fet i que m'han fet créixer com persona i gràcies per confiar en mi i gràcies per deixar-me la pel·lícula d'Amelie, també vull fer que les persones del meu voltant siguin felices.

A tots aquells companys del grau, que sense conèixer-me de res, el primer any de carrera em van ajudar, igualment que els meus professors de primer, que van entendre la meva situació, ja que sense tots ells no podria haver continuat degut els meus problemes de salut. A la meva àvia, que cada dia em portava de la universitat a casa i al revés pel mateix motiu.

A Neus i Jaume, els meus dos grans amics, per haver-hi estat i continuar estant.

Als meus germans, perquè tot i ser uns riallers, cabronets i guerrers, han sabut respectar les meves hores d'estudi i m'han abraçat aquells dies que ho he necessitat.

A Marc, per tantes hores d'estudi i complicitat al meu costat i fer-me somriure tot i l'estrès dels estudis.

A tots els professors del grau que m'han sabut guiar i orientar.

A Jordi Planes i Daniel Gibert, per confiar en mi, oferir-me aquest treball de final de grau, per orientar-me i ajudar-me i per valorar la meva feina. I a Ruben Martins, per ajudar-me quan he tingut dubtes i dedicar-me una mica del seu temps en la seva estada a Lleida.

I per últim, però no menys importants, als meus pares, Mar i Jaume, pels bons valors que m'han inculcat i perquè m'han donat ànims per seguir els meus somnis, perquè quan volia rendir-me no m'ho han permès, per recolzar-me en cadascuna de les meves bogeries, però, sobretot, fer-me sentir

la persona més estimada del món. Gràcies, espero que tingueu paciència amb mi, ja que no marxaré de casa fins a tenir el màster acabat.

Capítol 1

Introducció

Avui dia, segons el Consumer Barometer de Google [2], a Espanya un 87% de la població fa servir telèfons intel·ligents i un 41% utilitza tauletes a la seva rutina diària. Molts d'aquests dispositius fan servir tecnologies Android, de fet segons un article de Kantar [3] les vendes de l'any passat, 2017, d'Android superaven el 85%.

Amb els seus smartphones i tauletes la gent participa en un nombrós tipus d'activitats, des de consultar les xarxes socials fins a realitzar transaccions bancàries. Essent aquestes les seves utilitats, fan que la seguretat d'aquests dispositius sigui molt important. També s'ha de considerar que Android és el sistema operatiu que tenen més usuaris als seus mòbils i aquest factor crida als crackers a fer programari maliciós destinat a executar-se en aquest sistema operatiu.

A més dels aspectes anteriorment esmentats, el que fa més fàcil que hi hagi més apps malicioses a Android que a iOS (el sistema operatiu d'smatphone que fan servir més usuaris després d'Android) és que la botiga d'Android, la Play Store, és bastant menys segura que la d'Apple[4]. A la botiga d'Apple les aplicacions abans d'estar de cara al públic passen un procés més escrupolós, que controla més la qualitat i la seguretat de les aplicacions.

En el pronòstic de Malware 2018 de Sophos[5], es veu el programari

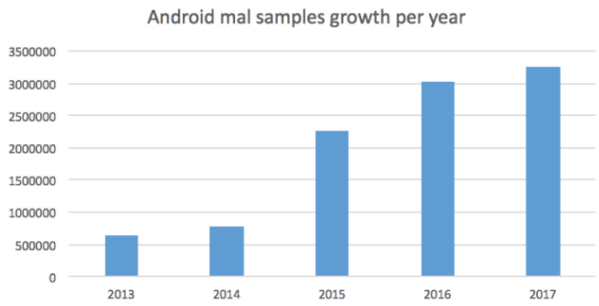


Figura 1.1: Creixement del malware a Android

maliciós d'Android que a dia d'avui està en marxa, amb ransomware com una amenaça cada vegada més gran. Segons l'anàlisi de SophosLabs, la quantitat d'atacs a clients de Sophos que fan servir dispositius Android va augmentar gairebé tots els mesos en 2017.

La quantitat d'aplicacions malicioses d'Android ha augmentat constantment en els últims quatre anys. En 2013, poc més de mig milió eren maliciosos. Per 2015 havia augmentat a poc menys de 2,5 milions. Per 2017, la quantitat era de gairebé 3,5 milions (fig. 1.1).

Tots els factors anomenats fan que sigui urgent trobar les millors formes de detecció de malware per a aplicacions del sistema operatiu de Google.

En aquest treball es pretén fer un estudi de l'ús de xarxes neuronals per a la classificació de programari maliciós del sistema operatiu de smartphones Android. Per fer-ho, es decideix construir un graf a partir de les crides i el flux de l'aplicació i buscar una forma de representar-lo de forma que una xarxa neuronal pugui entrenar amb grafs com entrada. El dataset que s'ha fet servir per fer les proves és Drebin[6], i internament per la construcció dels callgraph s'ha fet servir Apposcopy [7].

Capítol 2

Objectius

La part pràctica d'aquest Treball de Final de Grau és la implemantació d'una eina que classifiqui de malware d'Android, i comprovar l'eficàcia de les tecnologies emprades per a detecció de programari maliciós, concretament xarxes neuronals.

La idea del projecte sorgeix a partir dels articles d'Apposcopy [7] i Astroid[8], que través del callgraph d'una aplicació d'Android fan servir diferents tècniques (entre elles l'ús de MaxSat) per categoritzar una aplicació amb la seva família de malware d'Android corresponent.

L'objectiu principal d'aquest treball és treballar en el disseny i implementació d'un sistema de classificació automàtica de malware i estudiar l'eficiència de l'ús de callgraphs i xarxes neuronals per a la classificació de programari maliciós executable en Android.

Per la realització tot es basarà, principalment, en l'estudi de les crides a sistema de l'aplicació en el sistema Android. Es tindran en compte tant les crides entre diferents classes, com les crides a l'API d'Android que són sospitoses. Per sort, l'eina Apposcopy ajudarà en part del treball del sistema de detecció, construint el callgraph i el control taint flow graph.

Capítol 3

Estat de l'art

Donada la situació actual sobre els dispositius Android (molts usuaris realitzant transaccions de molts tipus i existeix poca seguretat al sistema operatiu), s'està fent molta investigació per trobar diferents solucions per millorar la seguretat d'Android. Algunes tècniques d'evaluació d'aplicacions que es fan servir són l'anàlisis estàtica i dinàmica i l'ús de machine learning, entre altres.

- Anàlisi estàtica: és una tècnica que avalua els comportaments maliciosos en el codi font, dades o fitxers binaris sense executar directament l'aplicació. La seva complexitat ha augmentat a causa de l'experiència que han adquirit els cibercriminals en el desenvolupament d'aplicacions i s'ha demostrat que és possible evitar-ho a partir d'tècniques d'obertura.[9]

- Anàlisi dinàmica: són mètodes que estudien el comportament del malware en execució mitjançant simulació de gestos; en aquesta tècnica s'analitzen els processos en execució, la interfície d'usuari, connexions de xarxa, obertura de preses, entre altres. Per altra banda, ja existeixen tècniques que permeten evadir el procés realitzat per l'anàlisi dinàmica, on el malware té la capacitat de detectar entorns sandbox i detectar el seu mal funcionament. [10]

- Ús de machine learning: Són molts els articles que es poden trobar sobre l'ús de machine learning per classificació, detecció i/o anàlisis de

malware [11][12][13][14][15][16][17] (sobretot per en l'entorn Android [18][8][19]), i són moltes les tècniques emprades.

Una de les raons per les quals aplicar xarxes neuronals a la visió per computador, el reconeixement de veu i el processament del llenguatge natural ha guanyat tan èxit és la seva capacitat d'aprendre funcions de dades en brut, com ara píxels o caràcters de text individuals. [20]

Inspirats en aquests èxits, un equip d'investigadors d'NVIDIA [20] va voler veure si una xarxa neuronal podria ser entrenada només amb els bytes bruts d'un arxiu executable per determinar si el fitxer és malware.

Si es pugés aconseguir això, es podrien simplificar molt les eines utilitzades per detectar malware, millorar la precisió de la detecció i identificar les característiques no obertes, però importants, que exhibeix el malware. NVIDIA va fer la seva implementació amb molt bons resultats (fig 3.1).

Train set	Test set	Our model AUC	Byte n-grams AUC	PE-Header Network
Group B train (small)	Group A	**98.5**	98.4	97.7
Group B train (small)	Group B test	95.8	**97.9**	91.4
Group B train (large)	Group A	**98.1**	93.4	–
Group B train (large)	Group B test	**98.2**	97.0	–

Figura 3.1: Rendiment de la implementació d'Nvidia d'un detector de malware amb xarxes neuronals en comparació amb els rendiments obtinguts pels millors detectors de malware en aquell moment.

Investigadors d'una companyia de Sophos, Invicea, també van experimentar l'ús de les xarxes neuronals per la detecció de malware[21].

Van implementar una eina formada pels components següents (fig.3.2): el primer component extreu quatre tipus diferents de funcions complementàries de les binàries estàtiques benignes i malicioses; el segon component és el classificador de xarxa neural profunda que consta d'una capa d'entrada, dues capes amagades i una capa de sortida; el component final és el calibrador de

puntuació, que tradueix els resultats de la xarxa neuronal a una puntuació que es pot interpretar de manera realista.

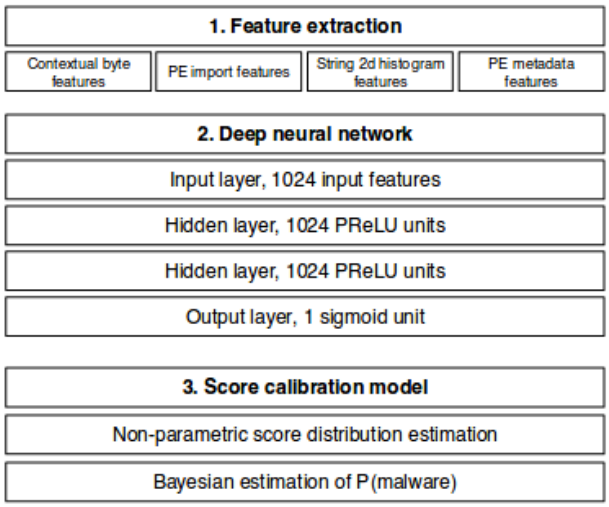


Figura 3.2: Arquitectura de l'eina basada en xarxes neuronals, per detecció de malare, implementada per Invicea.

Aquesta eina va aconseguir un índex de detecció del 95% i una taxa de falsos positius del 0,1% sobre un conjunt de dades experimentals de més de 400.000 binaris.

Capítol 4

Coneixements previs a la implementació

Per a la realització d'aquest treball de final de grau, ha sigut necessari repassar coneixements apresos durant el grau, igualment que adquirir-ne molts de nous, la majoria d'ells teòrics per entendre bé el context sobre el qual es treballaria. Durant els següents apartats es mostra un petit resum de cadascun dels coneixements apresos més destacables.

Ja que el tema principal del TFG és l'anàlisi de programari maliciós, el primer pas va ser la documentació teòrica sobre el malware, els seus tipus, anàlisi de malware i tipus d'anàlisi (apartat 4.2).

En segon lloc, com a segon tema principal del treball, l'estudi de l'entorn en el qual s'executa el malware a analitzar: Android. Sobre aquest sistema operatiu de mòbils es va fer recerca d'informació sobre els aspectes bàsics de les aplicacions, la seguretat vigent al sistema operatiu, etc. (apartat 4.1).

A més a més, i per comprovar que la recerca sobre detecció o classificació de malware d'aquesta plataforma és realment important, es van buscar alguns exemples reals i coneguts de programari maliciós d'Android, que haguéssim afectat a un nombre elevat de dispositius (apartat 4.3).

Ja que la part pràctica del TFG consisteix a treballar amb machine learning, la següent etapa va consistir a buscar informació sobre machine learning, xarxes neuronals i més concretament TensorFlow (apartat 4.4).

4.1 Android

Les eines de l'SDK d'Android compilen el codi juntament amb qualsevol fitxer de dades i recursos en un APK, un paquet d'Android. Un fitxer APK conté tots els continguts d'una aplicació d'Android i és el fitxer que utilitzen dispositius amb Android per instal·lar l'aplicació.

4.1.1 Seguretat

Cada aplicació d'Android viu a la seva pròpia caixa de seguretat, protegida per les següents característiques de seguretat d'Android:

- El sistema operatiu Android és un sistema Linux multiusuari en el qual cada aplicació és un usuari diferent.
- De manera predeterminada, el sistema assigna a cada aplicació un ID d'usuari exclusiu de Linux, que només coneix el Sistema Operatiu i és desconegut per l'aplicació. El sistema estableix permisos per a tots els fitxers d'una aplicació de manera que només l'identificador d'usuari assignat a aquesta aplicació pugui accedir-hi.
- Cada procés té la seva pròpia màquina virtual (VM), de manera que el codi d'una aplicació s'executa aïlladament d'altres aplicacions.
- De manera predeterminada, cada aplicació s'executa en el seu propi procés de Linux. El sistema d'Android inicia el procés quan cal executar qualsevol dels components de l'aplicació i, a continuació, tanca el procés quan ja no és necessari o quan el sistema ha de recuperar la memòria per a altres aplicacions.

El sistema d'Android implementa el principi de menys privilegis. És a dir, cada aplicació, per defecte, només té accés als components que requereix per

fer el seu treball i no a més. Això crea un entorn molt segur en què una aplicació no pot accedir a parts del sistema per a les quals no es dóna permís.

Tanmateix, hi ha maneres perquè una aplicació comparteixi dades amb altres aplicacions i que una aplicació accedeixi als serveis del sistema:[22]

- És possible fer que dues aplicacions comparteixin la mateixa ID d'usuari de Linux, en aquest cas poden accedir als fitxers de l'altre. Per conservar els recursos del sistema, les aplicacions amb el mateix ID d'usuari també poden organitzar-se en el mateix procés de Linux i compartir la mateixa màquina virtual. En aquest cas, les aplicacions també s'haurien d'iniciar amb el mateix certificat.

- Una aplicació pot demanar permís per accedir a dades del dispositiu, com ara els contactes de l'usuari, els missatges SMS, la targeta SD, la càmera i el Bluetooth. L'usuari ha de concedir explícitament aquests permisos. En versions anteriors a la 6.0, els permisos s'avisaven en fer la instal·lació de l'aplicació i l'usuari els havia d'acceptar tots. En les noves versions, en el moment que es necessita un nou permís es demana a l'usuari el permís d'accés en concret i l'usuari pot denegar o acceptar el permís.

Aquest últim aspecte millora la seguretat en les noves versions, respecte a les que s'havien d'acceptar tots els permisos per la instal·lació de l'aplicació. En moltes ocasions els usuaris no miraven els permisos quan es demanàvem tots de cop, i acceptaven. No els semblava estrany que, per exemple, una aplicació d'edició d'imatges demanés permisos per fer trucades.

4.1.2 Components d'una APP

Els components de l'aplicació són els blocs bàsics essencials d'una app d'Android.

Hi ha quatre tipus diferents de components:

- Activities: Punt d'entrada per interactuar amb l'usuari. Cada activitat representa una sola pantalla amb una interfície d'usuari.
- Services: Punt d'entrada de propòsit general per mantenir una aplicació executant-se en segon pla per tot tipus de motius. És un component que s'executa en segon pla per realitzar operacions de llarga durada o per realitzar treballs per a processos remots. Un servei no proporciona una interfície d'usuari.
- Receivers: Un receptor de difusió és un component que permet que el sistema lliuri esdeveniments a l'aplicació fora d'un flux d'usuari habitual, permetent que l'aplicació respongui als anuncis de difusió a tot el sistema. Atès que els receptors de difusió són una altra entrada ben definida a l'aplicació, el sistema pot transmetre emissions fins i tot a aplicacions que no s'estan executant actualment.
- Providers: Un proveïdor de contingut gestiona un conjunt compartit de dades de l'aplicació que podeu emmagatzemar al sistema de fitxers, en una base de dades SQLite, a la web o en qualsevol altra ubicació d'emmagatzematge persistent que pugui accedir a la vostra aplicació. A través del proveïdor de contingut, altres aplicacions poden consultar o modificar les dades si el proveïdor de contingut ho permet.

Cada tipus té un propòsit diferent i té un cicle de vida diferent que defineix com es crea i destrueix el component. [22]

4.1.3 Activació de components

Tres dels quatre tipus de components: activitats, serveis i receptors de difusió, s'activen mitjançant un missatge asincrònic anomenat intenció. Els intents s'uneixen a components individuals en temps d'execució. Es podem imaginar com a missatgers que sol·liciten una acció d'altres components, tant si el component pertany a l'aplicació com si és un altre.

Es crea una intenció amb un objecte Intent, que defineix un missatge per activar un component específic (intenció explícita) o un tipus específic de

component (intenció implícita).

Hi ha mètodes separats per activar cada tipus de component:

- Es pot iniciar una activitat amb un intent de startActivity() o startActivityForResult() (quan es vulgui que l'activitat retorni un resultat).
- Amb Android 5.0 (nivell API 21) i posterior, es pot utilitzar la classe JobScheduler per programar accions. Per a versions anteriors d'Android, es pot iniciar un servei passant un intent de startService(). Es pot enllaçar amb el servei passant un intent de bindService().
- Es pot iniciar una emissió passant una intenció a mètodes com sendBroadcast(), sendOrderedBroadcast() o sendStickyBroadcast(). [22]

4.1.4 AndroidManifest.xml

Document situat a l'arrel de l'APK. És un arxiu de configuració on estan escrites les configuracions bàsiques d'una APP.

Abans que el sistema d'Android pugui iniciar un component d'aplicació, el sistema ha de saber que el component existeix llegint el fitxer de manifest de l'aplicació, AndroidManifest.xml. Una aplicació ha de tenir declarats tots els components en aquest fitxer.

El manifest fa diverses coses a més de declarar els components de l'aplicació, com ara identificar els permisos que s'han de demanar a l'usuari, com accés a internet o als arxius del mòbil, declarar la versió mínima d'Android sobre la que pot córrer l'aplicació, declarar quines llibreries que no són d'Android s'han de fer servir, etc. [23]

4.2 Malware

Malware és l'abreviació de "Malicious Software"(programari maliciós). És programari desenvolupat amb l'objectiu de tenir accés als dispositius o causar danys en un dispositiu o una xarxa. Tot tipus de programari que danyi a

un usuari, un dispositiu o la xarxa es pot considerar programari maliciós. [24]

En aspectes generals, els tipus de malware més coneguts són els següents[1]: [25] [26] [27]

- Virus: afecten els arxius de programes i arxius personals.
- Spyware: software que recompila informació personal.
- Worm: malware que pot replicar-se per la web.
- Trojan horse: programa que es veu, i inclús pot funcionar com un programa legítim.
- Browser hijacker: software que modifica navegadors web.
- Rootkit: software que obté drets d'administrador amb intencions malicioses.
- Malvertising: l'ús de publicitat legítima en línia per difondre software maliciós.
- Ransomware: pren el control del sistema i demana un rescat a l'usuari per tornar-li a donar el control de dispositiu.
- Backdoor: s'instal·la al dispositiu per permetre accedir a l'atacant, així l'atacant pot entrar sense autentificació i pot executar comandes al sistema local.
- Botnet: és semblant al backdoor, però en tots els dispositius infectats amb un malware d'aquest tipus s'executen les comandes que l'atacant crida des d'un servidor.
- Dowloaders: malware que només serveix per instal·lar més malware. Normalment l'atacant instal·la els downloaders quan té accés al sistema.

[1] Alguns d'aquests termes també es fan servir, en alguns contextos, per parlar de goodware

- Information-stealing malware: aquest malware recull informació de l'ordinador de la víctima i l'envia a l'atacant. En molts casos es fa servir per robar contrasenyes.
- Launcher: software maliciós per llançar altres programaris maliciosos.
- Scareware: aquest malware es fa passar per un software segur, de l'estil d'un antivirus, i mostra a l'usuari missatges com que està desprotegit i per protegir-se ha de comprar un software (que representa ser un antivirus més potent) per estar segur, quan realment el software comprat no fa més que esborrar en scareware.
- Spam-sending malware: malware que infecta la màquina de la víctima i després fa servir la seva màquina per enviar spam.

El malware pot infectar un dispositiu de diverses formes. La majoria de vegades, el malware l'instal·len directament els usuaris, amb programes que contenen software maliciós, sense que els usuaris siguin conscients, per accident.

4.2.1 Malware analisys

El malware analisys és l'art de disseccionar el programari maliciós per entendre com treballa, com identificar-lo i com derrotar-lo o eliminar-lo. Aquest procés es realitza ja una vegada trobat el malware.

Hi ha quatre tipus d'anàlisis de malware:

- Anàlisi estàtica bàsica: consisteix a examinar el fitxer executable sense fixar-se en les instruccions actuals. Aquest tipus d'anàlisi pot confirmar si un fitxer és maliciós, si dóna informació sobre la seva funcionalitat. L'anàlisi estàtica bàsica és ràpida i senzilla, però és molt poc efectiva contra malware sofisticat i pot passar per alt comportaments importants del software que s'està analitzant.
- Anàlisi dinàmica bàsica: consisteix a córrer el malware i observar el seu comportament al sistema. Abans de córrer el programari s'ha de

fer un sistema segur i controlat al qual es pugui estudiar l'execució sense riscos de fer malbé el dispositiu o la xarxa. Igualment que amb les tècniques d'anàlisi estàtica bàsica, aquest tipus d'anàlisi es pot implementar sense gaires coneixements de deep programming, però no és efectiu amb tots els malwares i es pot perdre informació sobre les funcionalitats del programari maliciós.

- Anàlisi estàtica avançada: consisteix a fer enginyeria inversa del programari maliciós, carregant l'executable en un desassemblador i mirant les instruccions del programa per descobrir què fa el programa. Les instruccions són executades per la CPU, així que aquest tipus d'anàlisis diu exactament què fa el programa. Adiferència de l'anàlisis estàtica bàsica, l'anàlisi estàtica avançada té una corba d'aprenentatge més pronunciada i requereix coneixement especialitzat sobre desassemblatge, construcció de codi, conceptes del sistema operatiu pel qual està pensat el malware.

- Anàlisi dinàmica avançada: fa servir el debuggador per examinar l'estat intern d'una execució de malware. Les tècniques d'aquest tipus d'anàlisi proporcionen una altra forma d'extraure informació detallada d'un executable. Aquestes tècniques són més útils quan s'intenta obtenir informació que és difícil de trobar amb altres tècniques.

Hi ha tres casos d'ús habituals en els que s'aplica l'anàlisi de malware:[27]

- Gestió d'incidències de seguretat informàtica: si una organització sospita que poden tenir un malware als seus sistemes, l'empresa pot desitjar realitzar una anàlisi de malware per determinar si realment és programari maliciós i llavors calcular l'impacte d'aquest programari sobre els sistemes.

- Investigació de malware: els investigadors acadèmics de malware, poden realitzar anàlisis de malware per comprendre com es comporta el programari maliciós i trobar les últimes tècniques que es fan servir.

- Indicador d'extracció de compromís: els proveïdors d'antivirus i semblants, realitzen anàlisis de malware en bloc per determinar possibles

nous indicadors de compromís (informació que pot alimentar el producte de seguretat o la solució per ajudar a les organitzacions a defendre's millor contra els atacs de malware).

4.3 Malware en aplicacions Android

Aquesta secció és per ensenyar alguns casos reals de programari maliciós d'Android coneguts, per demostrar que és una realitat l'existència de malware en aquest sistema operatiu.

4.3.1 Cas "Gooligan"

Més d'un milió de comptes de Google van ser infectades amb aquest malware d'Android.

Aquesta amenaça atacava a dispositius que instal·laven aplicacions de tercers, que no estan disponibles a la Play Store, la botiga oficial d'Android.

Gooligan primer rooteja[2] els dispositius mòbils, i una vegada està rootejat, instal·la un software que roba la informació dels comptes de Google vinculats al dispositiu. la infromació robada poden anar des de les dades personals de l'usuari fins a fitxers del Google Drive o Google Fotos. Aquesta informació és enviada a un servidor.

Per sort, a les últimes versions d'Android, a partir de la versió 6.0, els mòbils estan segurs contra aquest malware.

Googligan va afectar un 57% dels comptes d'Àsia, un 19% a les americanes, un 15% a Àfrica i a un 9% dels comptes europeus. [28]

[2]El rooting es duu a terme generalment amb l'objectiu de superar les limitacions que els operadors de telefonia mòbil i els fabricants de maquinari col·loquen en alguns dispositius, tenint com a resultat la capacitat de fer coses que un usuari sense root no pot fer com, per exemple, desinstal·lar les aplicacions per defecte i canviar-les per unes altres.

4.3.2 Cas del "virus de la policia"

Malware del tipus ramsomware 4.2.

Aquest virus als seus inicis només afectava ordinadors, però desde l'any 2016 també afecta dispositius mòbils.

El que destaca a aquest malware és que entra al dispositiu i en un moment qualsevol, a l'usuari li apareix una pantalla amb un missatge que sembla ser de la policia del país on es troba, amb informació detallada, com dades personals o la ubicació, per semblar més real.

La pantalla indica que si no es paga una multa en un temps determinat, el dispositiu es mantindrà bloquejat. Durant aquest temps l'usuari no pot accedir als seus arxius, a la vegada que no pot interactuar de cap forma amb el mòbil fins que es paga el rescat. Si no es paga la multa en aquell termini, realment no hi ha forma de tornar a fer servir el dispositiu amb normalitat.

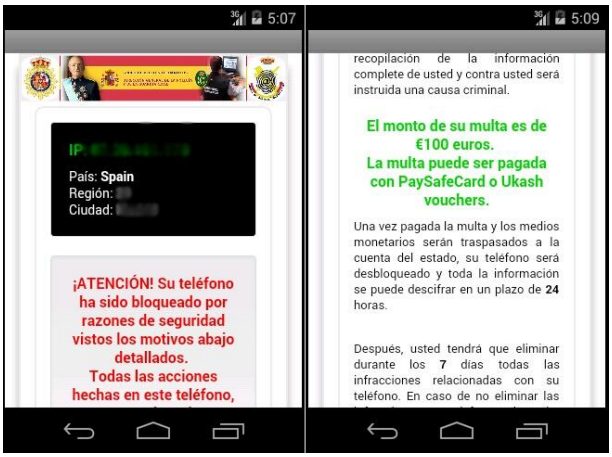


Figura 4.1: Missatge d'amença mostrat pel "virus de la policia"

Per sort, mentre que a la versió d'ordinador es xifren totes les dades emmagatzemades a l'ordinador i l'única forma de recuperar-les és pagant a temps, a la versió del malware a Android el que es fa és bloquejar el mòbil fins que es faci el pagament. Així doncs, encara es poden recuperar les dades

connectant el dispositiu Android a un ordinador i copiant les dades, però per tornar a fer que el mòbil sigui útil s'ha de formatar.

L'aplicació maliciosa s'instal·la al mòbil juntament amb una aplicació que sembla inofensiva.[29]

4.3.3 El cas "AdultSwine"

Malware del tipus scareware. Google va haver d'eliminar a la Play Store més de 30 aplicacions, que semblaven ser jocs dirigits a nens, que contenien aquest malware. Aplicacions amb aquest malware han estat descarregades més de tres milions de vegades.

Qui va crear aquest malware va fer-ho sota el nom de creadors habituals de jocs per distribuir el seu programari maliciós i fer diners.

Una vegada descarregades, les aplicacions mostraven anuncis en forma de "pop-up"amb contingut pornogràfic, i invitaven als usuaris a descarregar-se una aplicació de seguretat falsa per eliminar el"virus"que generava els anuncis. En alguns casos també oferia serveis prèmium falsos, sense valor.[30]

4.3.4 El cas "LOAPI"

Malware moltipropòsit que permet als atacats minar la criptodivisa "Monero" , llençar atacs DDoS[3] o subscriure a les víctimes en serveis de pagament.

Mitjançant campanyes de missatges SMS, publicitat i altres tècniques d'spam, Loapi va aconseguir distribuir-se entre un gran nombre de víctimes.

Després d'instal·lar l'aplicació fraudulenta el malware tracta d'escalar permisos en el dispositiu, demanant permisos d'administrador en bucle fins que

[3]DDoS són les sigles de Distributed Denial of Service. La traducció és "atac distribuït denegació de servei", i traduït de nou vol dir que s'ataca al servidor des de molts ordinadors perquè deixi de funcionar.

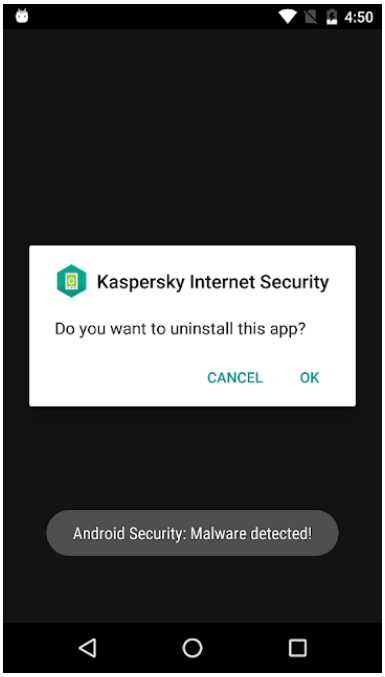


Figura 4.2: Missatge fraudulent per desinstal·lar un AV legítim.

l'usuari els accepta. Posteriorment, Loapi comprova si el dispositiu està 'rootejat' (encara que aquesta versió no fa ús en cap moment d'aquests privilegis).

Després d'obtenir els permisos necessaris s'amaga en el sistema, bé ocultant la icona al menú o simulant ser una aplicació d'antivirus.

El troià a més implementa mecanismes per protegir-se i evitar ser desinstal·lat: pot actualitzar de forma remota una llista negra d'aplicacions que podrien suposar una amenaça i enganyar l'usuari perquè siguin desinstal·lades. A més de tancar la finestra d'e configuració per evitar que l'usuari pugui revocar els permisos (fig. 4.2).

Entre les diferents accions que es poden dur a terme amb aquest malware, cal destacar l'opció de minat de criptomonedes, que podria deixar el dispositiu Android inutilitzat a causa de l'ús tan intensiu de CPU.[31][32]

4.3.5 Famílies de malware Android

Les famílies de malware d'Android consisteixen en grups d'aplicacions malware que són variants d'una principal o tenen comportaments molt similars entre elles. Els següents apartats estan dedicats a les tres famílies en les quals BadsDroid (la part pràctica d'aquest treball) sabrà classificar el malware.

"DroidKungfu"

DroidKunfu és el primer troià malware d'Android que aconseguia els antivirus dels dispositius, trobat el 2011.

El seu funcionament és l'habitual que el d'altres troians: és instal·lat al dispositiu perquè, de cara a l'usuari, sembla un programari legítim.

Una vegada dintre del dispositiu, aquest malware intenta rootejar el mòbil i així aconseguir diverses dades del dispositiu:

- Identitat de l'equip mòbil internacional (IMEI).
- Model de dispositiu mòbil.
- Operador de xarxa.
- Tipus de xarxa.
- API del sistema operatiu.
- Tipus de sistema operatiu.
- Informació emmagatzemada a la memòria del telèfon.
- Informació emmagatzemada a la memòria de la targeta SD.

Aquestes dades el malware les emmagatzema i les envia a un servidor remot.

Totes les aplicacions variants d'aquest malware originari, estan dintre de la família DroidKunfu, i són molt habituals.

“OpFake”

Les aplicacions d'aquesta família són de tipus trojan horse i totes són variants de l'Opfake original.

OpFake és el segon Trojan-SMS més popular anomenat Opera Mini Mobile Browser per ser una descàrrega falsa. Opfake utilitza el mètode d'instal·lació de repacking i envia SMS a telèfons de costos especials amb dades de la SIM, descarrega altres aplicacions malicioses i les emmagatzema a la targeta SD. L'atacant primer crea un lloc web fals per atraure als clients que descarreguin l'Opera Mini Browser amb els noms dels fitxers apk semblants al de l'Opera. [33]

“FakeInstaller”

El malware d'aquest tipus és dels més habituals dels que es coneixen. Més del 60% de les mostres d'Android processades per McAfee són FakeInstallers.

FakeInstaller envia missatges SMS als números de tarifació premium de propietat de l'atacant, sense el consentiment de l'usuari, passant-se com a instal·lador d'una aplicació legítima. Hi ha moltes variants per aquest malware i es distribueix en centenars de llocs web i mercats falsos. La propagació d'aquest malware augmenta cada dia.

L'engany comença quan els usuaris busquen una aplicació popular i accedeixen a un fals lloc oficial o mercat fals a través de motors de cerca o xarxes socials. Les aplicacions semblen ser legítimes, incloses captures de pantalla, descripcions, comentaris d'usuaris, vídeos, etc. Les víctimes cauen en la trampa de descarregar i instal·lar el programari maliciós. Quan s'executa FakeInstaller, mostra un acord de servei que indica a l'usuari que s'enviaran un o més SMS; aquest acord s'ha trobat en rus o anglès.

La interfície pot ser confusa. L'usuari es veu obligat a fer clic a un botó d'acord o següent, que envia els missatges SMS premium. També s'han vist

versions que envien els missatges abans que la víctima faci clic en un botó.[34]

4.4 Machine Learning

La intel·ligència artificial és una de les branques de la Informàtica, amb fortes arrels en altres àrees com la lògica i les ciències cognitives.

Podem dividir la intel·ligència artificial dos grans grups: la IA robusta i la IA aplicada.

- Intel·ligència Artificial robusta o Strong AI: tracta sobre una intel·ligència real en què les màquines tenen similar capacitat cognitiva que els humans, cosa que, com els experts prediuen, encara queden anys per assolir.
- Intel·ligència Artificial aplicada o Weak AI (Narrow AI o Applied AI): aquí és on entren l'ús que fem a través d'algoritmes i aprenentatge guiat amb el Machine Learning i el Deep Learning. La idea bàsica de qualsevol tasca d'aprenentatge automàtic és formar el model, basat en algun algoritme, per realitzar una determinada tasca: classificació, clusterització, regressió, etc. l'entrenament es fa a partir del conjunt de dades d'entrada i el model que es construeix s'utilitza posteriorment per fer prediccions.

Machine Learning és la ciència (i l'art) de programar ordinadors perquè puguin aprendre de les dades.

Consisteix de 5 pasos indicats a la figura 4.3 [35]:

1. El set de dades es carrega des d'un arxiu i es guarda en memòria.
2. Les dades es netegen, transformen i normalitzen per a ser adequades per l'algoritme. A més a més les dades es separen en dos: en "trainig set" i "validation set".
3. Es construeix el model amb l'algoritme triat i el "training set".

4. El model entrenat es prova amb el "validation set". Els resultats es fan servir per crear altres models.

5. Es tria el millor model.

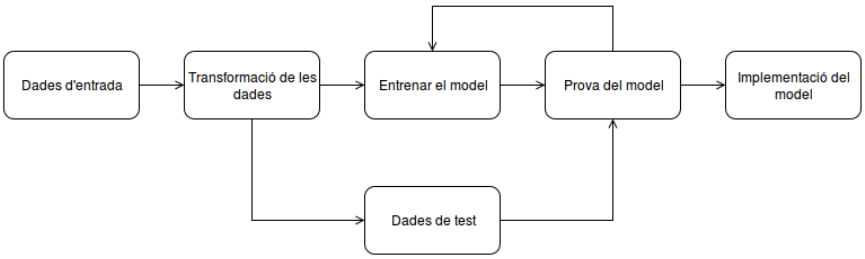


Figura 4.3: Flux de treball general del procés d'aprenentatge automàtic

Hi ha tres gras categories d'aprenentatge:

- Supervised learning: depèn de dades prèviament etiquetades.
- Unsupervised learning: les dades no tenen etiquetes, però el programa rep una gran quantitat de dades amb les característiques pròpies d'un objecte, perquè pugui determinar què és a través de la informació recopilada.
- Reinforcement learning: la màquina aprèn en base de proves i errors. Encara que la màquina coneix els resultats des del principi, no sap quines són les millors decisions per arribar a obtenir-los. El que passa és que l'algoritme progressivament va associant els patrons d'èxit, per repetir-los una vegada i una altra fins perfeccionar-los.[36][37][38]

4.4.1 Xarxes Neuronals

Per entendre què és una xarxa neuronal primer s'ha d'entendre que és una neurona artificial o perceptró.

Neurona perceptró

Els perceptors van ser desenvolupats en els anys cinquanta i seixanta pel científic Frank Rosenblatt. Avui en dia, és més comú utilitzar altres models de neurones artificials, el tipus principal de neurones utilitzat és l'anomenada neurona sigmoide.

Un perceptró és una neurona que rep diverses entrades binàries i la seva sortida és també un valor binari.

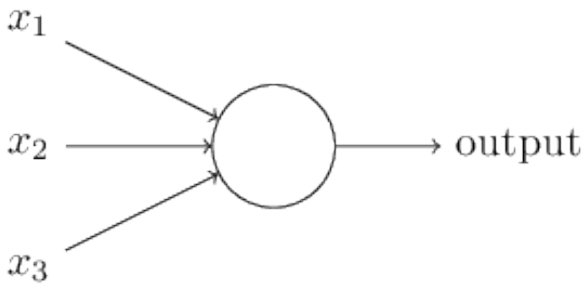


Figura 4.4: Representació gràfica d'un perceptró

Rosenblatt va introduir pesos, w1, w2, ..., nombres reals que expressaven la importància de les entrades respectives a la sortida. La sortida de la neurona, 0 o 1, es determina si la suma ponderada dels pesos i entrades és menor o major que un valor de llindar. Igual que els pesos, el llindar és un nombre real que és un paràmetre de la neurona.

Al variar els pesos i el llindar, podem obtenir diferents models de presa de decisions i amb una xarxa de perceptrons es poden fer decisions més complicades.

Neurona sigmoide

Les neurones sigmoide es representen de la mateixa forma que els perceptrons (fig. 4.4).

Igual que la neurona perceptró, la neurona sigmoide té entrades i sortides, però la diferència és que els seus valors no són valors binaris ni enters, són amb coma flotant.

La sortida d'una neurona sigmoide en comptes de ser una suma ponderada serà $\sigma(wx + b)$ on w és el pes, x les entrades i b, la bias. La bias és el que abans s'havia denominat *llindar*.

La funció que representa sigma és la següent:

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}}. \tag{4.1}$$

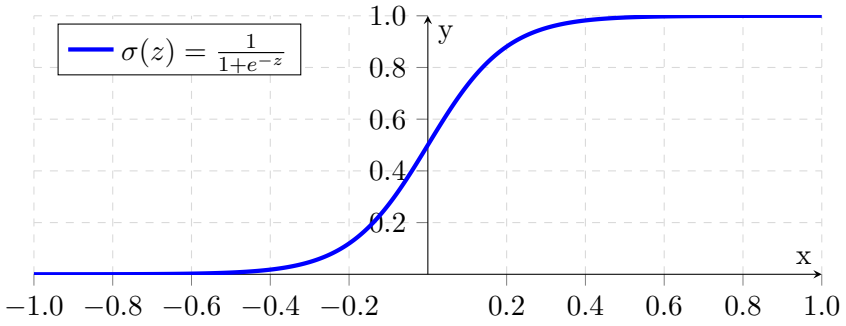


Figura 4.5: Representació de la funció sigma

D'aquesta forma, quan z = wx + b és gran i positiu, la sortida de la neurona sigmoide és aproximadament 1, tal com hauria estat un perceptró. Al cas contrari, quan el resultat de z = wx + b és molt negatiu, la sortida la sigmoide és aproximadament 0. Com veiem aquesta neurona actua de forma semblant al perceptró però no passa de zero a 1 de cop, com fa el perceptró.

Arquitectura

Una xarxa neuronal està composta per capes, les quals estan compostes per neurones. En les xarxes neuronals no convolucionals les neurones d'una capa estan connectades amb totes les neurones de la següent capa.

Normalment una xarxa neuronal té més de tres capes. La primera capa és la d'entrada i l'última, la de sortida. Totes les capes que es troben entre les dues anteriorment mencionades són les capes ocultes (hidden).

La sortida d'una neurona d'una capa que no sigui la de sortida, és l'entrada de les neurones de la següent capa, i cada sortida té un pes associat.

Cada una de les capes té una bias associada. Moltes vegades en esquemes dibuixats de xarxes neuronals la bias és una neurona més de la capa, que la seva entrada sempre és 1.

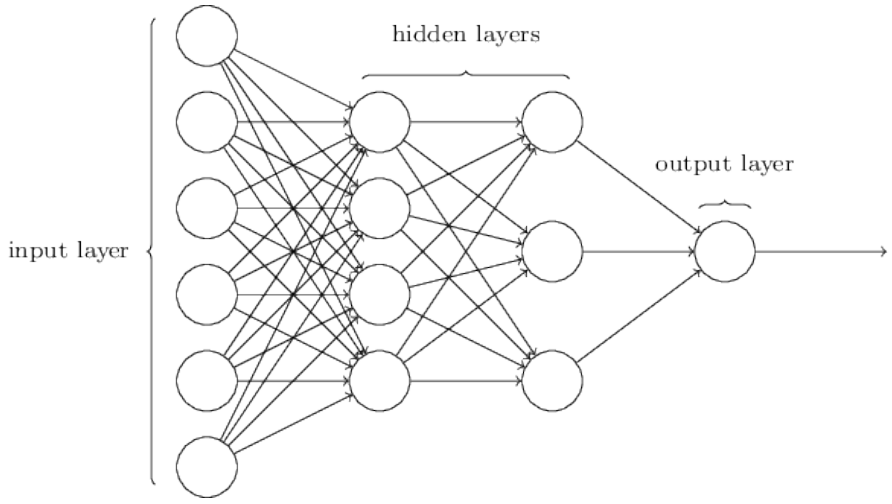


Figura 4.6: Esquema de l'arquitectura d'una xarxa neuronal [1]

Tot i que normalment les xarxes neuronals es representen com a la figura 4.6, poden tenir més d'una sortida. En el cas de tenir una sortida, com el valor de la sortida normalment oscil·la entre 0 i 1, si és més gran de 0.5 la sortida és positiva i en cas contrari, és negativa. En el cas de tenir més d'una sortida (habitual en xarxes neuronals de classificació), la sortida que s'activarà, la de la posició on es classificaria, segons els valors de l'entrada, serà la de valor més alt.[14][1]

4.4.2 Tensorflow

TensorFlow està dissenyat per a la formació i la inferència distribuïdes a gran escala, però també és prou flexible per donar suport a l'experimentació amb nous models d'aprenentatge automàtic i optimitzacions del sistema.

L'arquitectura principal de TensorFlow està reflectida a la figura 4.7.

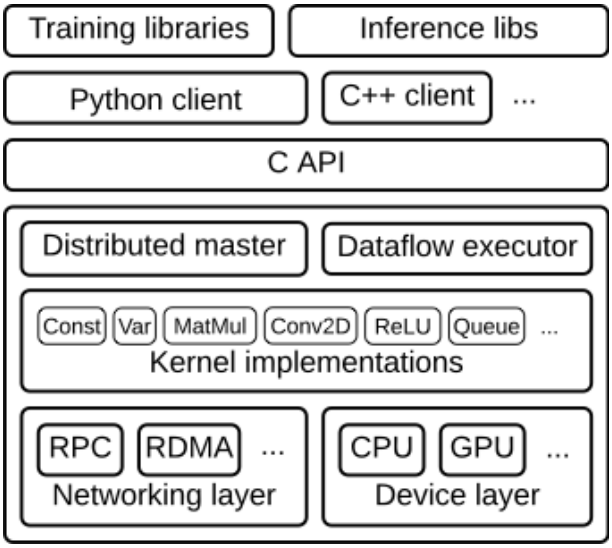


Figura 4.7: Arquitectura principal de TensorFlow

Els usuaris escriuen el programa TensorFlow del client que construeix un gràfic de computació. Aquest programa pot compondre directament operacions individuals o utilitzar una biblioteca de conveniència com l'API Estimadors per compondre capes de xarxa neurals i altres abstraccions de nivell superior.

El client crea una sessió, que envia la definició del gràfic al mestre distribuït com a buffer de protocol tf.GraphDef. Quan el client avalua un node o nodes al gràfic, l'avaluació activa una crida al mestre distribuït per iniciar la computació.

El temps d'execució conté més de 200 operacions estàndard, incloses

la manipulació matemàtica, la manipulació de matrius, el flux de control i les operacions de gestió de l'estat. Cadascuna d'aquestes operacions poden tenir implementacions del kernel optimitzades per a diversos dispositius. Molts dels kernels operatius s'implementen amb Eigen :: Tensor, que utilitza plantilles C++ per generar codi paral·lel eficient per CPU i GPU multicor.[39]

Capítol 5

BadsDroid

La part pràctica d'aquest TFG ha consistit en l'estudi de fer servir machine learning (en aquest cas, xarxes neuronals) per a la classificació de malware. Per fer-ho s'ha implementat una eina, sota el nom *BadsDroid*. Amb aquesta eina podem obtenir la precisió a l'entrenar una xarxa neuronal amb representacions d'aplicacions Android que es comenten més endavant [5.3.5] per classificar les apps en les famílies de malware a les que pertanyen. Per fer-ho BadsDroid fa una anàlisis estàtic anançada de les APKs Android.

5.1 Tecnologies i eines utilitzades

Per la realització de BadsDroid s'han fet servir nombroses tecnologies:

- Apposcopy: tot i que el propòsit d'Apposcopy és també la classificació de malware, aquesta eina únicament s'ha fet servir per a la generació dels callgraphs juntament amb el graf del flow de l'aplicació. Aquesta eina està explicada amb més detall a la secció 5.2
- Python: el llenguatge en el qual s'ha implementat és Python. Com BadsDroid és una eina per a fer recerca i en un començament tenia uns requisits molt fluixos, s'ha decidit implementar-la amb aquest llenguatge, ja que és un llenguatge amb el qual és còmode i ràpid programar (per si s'havien de fer canvis o començar de zero durant la implementació del projecte).

- Node2Vec: és un marc algorítmic per a l'aprenentatge de representació de grafs. Donat qualsevol gràf, pot aprendre representacions de funcions contínues per als nodes, que després es poden utilitzar per a diverses tasques d'aprenentatge automàtic. Aquesta eina s'ha utilitzat per crear els embeddings que representen cada un dels nodes d'un graf.

- Pycharm: a vegades els IDEs utilitzats es menyspreen i també s'han d'anomenar, ja que faciliten molt la feina. PyCharm és un entorn de desenvolupament integrat (IDE) utilitzat en la programació, específicament per al llenguatge Python. Està desenvolupat per l'empresa txeca JetBrains. Proporciona anàlisi de codi, un depurador gràfic, un integrador integrat de proves, integració amb sistemes de control de versions (VCSes) i se li poden instal·lar més plugins per facilitar encara més algunes altres coses.

- Git: per al control de versions i no perdre canvis anteriors que poguessin tornar a interessar en un futur. És un sistema de control de versions per fer el seguiment dels canvis en els fitxers informàtics. S'utilitza principalment per a la gestió del codi font en el desenvolupament de programari, però es pot utilitzar per fer un seguiment dels canvis en qualsevol conjunt d'arxius.

- GitHub: s'ha utilitzat simultàniament amb git per a controlar els canvis de l'aplicació i a més per tenir el programari en algun lloc més a part d'un ordinador físic, com a prevenció per si passava quelcom. És un servei d'allotjament web per al control de versions que utilitza Git. S'utilitza principalment per al codi informàtic. Ofereix tota la funcionalitat de control de versions distribuïdes i codi font de Git, a més d'afegir funcions pròpies. Proporciona control d'accés i diverses funcions de col·laboració, com ara el seguiment d'errors, les peticions de funcions, la gestió de tasques i els wikis per a cada projecte.

- TensorFlow: per a implementar i entrenar la xarxa neuronal que finalment classifica el malware en famílies. TensorFlow és una biblioteca de programari de codi obert per a la computació numèrica d'alt rendiment. La seva arquitectura flexible permet un fàcil desplegament

de la computació a través d'una varietat de plataformes (CPU, GPU, TPU) i des d'escriptoris fins a clústers de servidors a dispositius mòbils. Originalment desenvolupat per investigadors i enginyers de l'equip de Google Brain dins de l'organització AI de Google, compta amb un fort suport per a l'aprenentatge automàtic i l'aprenentatge profund i el nucli de computació numèrica flexible s'utilitza en molts altres dominis científics.

- Drebin: drebin comparteix un repositori de 5000 exemples d'apks Android. Aquests exemples s'utilitzen per entrenar BadsDroid.

A més d'aquestes tecnologies, en un començament s'havien fet servir altres, ja que als inicis s'estava implementant una eina pròpia per generar els callgraphs de BadsDroid, i així no dependre d'Apposcopy:

- Jadx: per deassemblar el codi de l'APK i obtenir els arxius .java des de un arxiu .APK. [40]

- APKTool: és una eina per a enginyeria inversa per aplicacions blanques tancades i d'Android. Es pot descodificar els recursos a la forma gairebé original i reconstruir-los després de fer algunes modificacions. Es feia servir per descodificar l'arxiu AndroidManifest.xml, d'on s'obté moltíssima informació de cara a fer el CallGraph d'una APK.

5.2 Apposcopy

Analitzador de malware basat en la semàntica per identificar una classe predominant de malware d'Android que provoca la informació privada de l'usuari. Apposcopy incorpora un llenguatge d'alt nivell per especificar signatures que descriuen característiques semàntiques de famílies de malware i una anàlisi estàtica per decidir si una aplicació donada coincideix amb una signatura de malware.

L'algoritme de concordança de signatura d'Apposcopy utilitza una combinació d'anàlisi de tinció estàtica i una nova forma de representació de programes anomenada Inter-Component Call Graph (fig. 5.1) per detectar

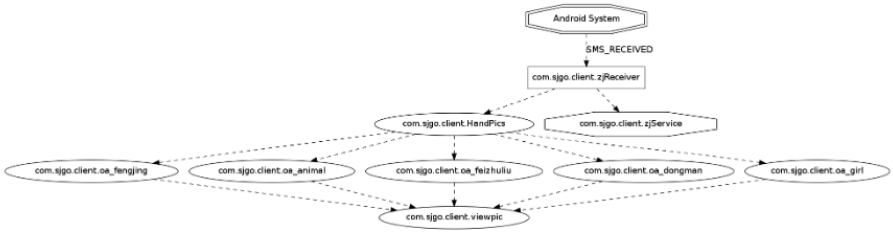


Figura 5.1: ICCG parcial per a una instància de la família de malware GoldDream

eficientment les aplicacions d'Android que tenen certes propietats de control i de flux de dades.[7]

Aquesta eina és utilitzada per la generació Inter-Component Call Graph (ICCG) i obtenir el graf de flux de les aplicacions.

5.3 Implementació

5.3.1 Primera versió

En un començament, tenint present Apposcopy i Astroid, s'anava a fer una eina molt semblant, on, a partir de l'APK de les aplicacions Android, es construiria un graf de crides per, en un futur, classificar el programari com a maliciós o no i classificar-lo en la seva família fent servir MaxSat.

La implementació de l'eina va començar implementant la part del generador del CallGraph. Per fer-ho, la primera part va ser sobretot d'adquirir informació teòrica sobre Android i sobre com es criden les classes internament entre elles. Després d'una primera intuïció i hipòtesis del funcionament per cridar classes de tipus diferents en Android, es va provar (utilitzant una mica prova i error, ja que la informació trobada a vegades era insuficient) si era com semblava.

Es va programar una eina que llegia del AndroidManifest.xml les classes que es feien servir i d'aquestes classes es parsejava el codi JAVA, obtingut al

fer servir aplicacions d'enginyeria inversa per decompilar l'apk, per trobar les crides. Segons les crides que s'anaven trobant es construïa un graph de crides. Finalment es comparava amb el CallGraph resultant d'Apposcopy per a la mateixa APK.

Posteriorment de la implementació d'una eina capaç de generar CallGrpahs i abans de començar a pensar en MaxSat, es va decidir fer servir xarxes neuronals, en comptes de MaxSat.

5.3.2 Segona versió

A partir d'aquest moment algunes idees que estaven presents canviarien. El següent pas abans de continuar amb la implementació era la recerca sobre el funcionament de les xarxes neuronals, per entendre-les i així fer que BadsDroid li donés com input la informació en la millor estructura de dades perquè poguessin servir com entrada a la xarxa neuronal.

A partir d'aquí les opcions eren múltiples. Es va optar per fer servir l'eina node2vec per representar cadascun dels nodes del graf resultant d'una aplicació. D'aquesta forma i com s'ha fet en altres ocasions en problemes similars (entrenar una xarxa neuronal amb "grafs" com entrada) el que es plateja fer és:

1. Crear el graf (que posteriorment es decideix extreure'l d'Apposcpy en comptes de crear-lo).
2. Realitzar camins aleatoris pel graf, sense repetir arestes, perquè no existeixi la possibilitat d'entrar en bucles infinits. Aquests camins seran tots d'una mida concreta, i si algun camí es queda sense sortida abans d'arribar a aquest nombre de passos, s'omple amb zeros l'espai restant.
3. Fer un nombre determinat de camins com el del pas anterior, per obtenir una matriu de dimensions MxN.
4. Substituir cadascun dels valors de la matriu per la seva representació de Node2Vec.

A partir d'aquesta implementació de BadsDroid, en el moment d'implementar-lo apareixen problemes que s'han d'afrontar i solucionar.

5.3.3 Tercera versió

El primer problema que es presenta és passant un mateix graf al Node2Vec diverses vegades. Els resultats a cada vegada són diferents, i això no convé. Es proven de fer algunes execucions i s'observa que, tot i variar, en la majoria de casos els valors són similars (no iguals) entre ells. Es decideix provar d'executar unes quantes vegades i fer la mitjana aritmètica dels valors de cada posició. D'aquesta forma el resultat obtingut, per un mateix graf, era sempre igual.

Durant aquest temps, encara hi ha dubtes "teòrics" sense resoldre. Una classe té un nom definit pel programador, però per facilitar la tasca a la xarxa neuronal, es vol que classes similars tinguin un nom similar, per facilitar la feina al Node2Vec i a la xarxa neuronal.

Android té quatre tipus de classes, i als callgrafs només es tenen en compte tres: *activities*, *services* i *receivers*.

Es decideix que, com l'important és saber quin tipus de classes es comuniquen amb altres, s'associarà un identificador per cada tipus (p.e. activities:1, receivers:2, services:3). Llavors el resultat és un graf amb nodes etiquetats únicament amb tres valors. En aquest graf sembla faltar informació important i es comenta de buscar tipus d'activitats i fer més subgrups.

5.3.4 Quarta versió

Just abans de començar amb l'implemntació ideada a l'última versió, s'aprofita l'estada a Lleida de Ruben Martins, per parlar amb ell sobre el tema. Comenta que BadsDroid només construeix un Callgraf, menetre que Apposcopy té en compte el "flow" de l'aplicació. Amb flow o flux es refereix a tipus de crides que fa servir per comunicar les classes i tractament d'algunes crides o permisos que ja es sap que podem ser sospitosos.

Ruben Martins recomana algunes aplicacions com FlowDroid, o inclús recomana fer servir Apposcopy directament.

A Apposcopy, es fa servir un graf amb arestes etiquetades i per BadsDroid no interessa que el graf sigui així. Aquesta mateixa sensació va tenir el Ruben Martins i es parla amb ell sobre el problema i finalment es conclou amb convertir el que a Apposcopy serien etiquetes de les arestes, en nodes que connecten els dos nodes que abans estaven connectats per l'aresta etiquetada en questió.

Després d'examinar l'eina FlowDroid i la seva sortida, es decideix fer servir Apposcopy per obetenir el flux de l'aplicació, ja que ja es coneix l'eina d'haver treballat als primers passos amb ella. A partir d'aquest moment es pot implementar tot BadsDroid, a falta de l'entrenament de la xarxa neuronal.

5.3.5 Cinquena versió

En la impementació d'aquesta versió es comença tractant les dades resultants del graf d'apposcopy.

El primer que es fa és consultar l'API d'Android, i per cadascun dels mètodes de l'API associar-li un número, que representarà aquell mètode al graf de BadsDroid.

Apposcopy extreu un json amb diversa infomació:

- ICCG: El callgraph de l'aplicació, que és equivalent al que BadsDoid generava en les primeres versions. La forma de cada aresta del ICCG és la següent:

```
{
    "val0":"com.xxx.yyy.MyBoolService",
    "val1":"com.xxx.yyy.AlarmReceiber"
}
```

on la part que es mostra després del dos punts és el nom d'una classe de la aplicació Android. Aquestes classes sempre són del tipus activity, service o receiver. Es representen al graf de BadsDroid com arestes direccionades (fig. 5.2).

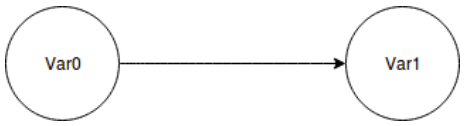


Figura 5.2: Representació gràfica de com es representen les connexions entre classes a BadsDroid

- IntentFilters: són els filtres d'intencions que hi ha a l'aplicació. Aquest component es fan servir per indicar quina informació del sistema es necessita a l'aplicació. Igual que al ICCG, es mostra en parelles:

  ```
  {
  "val0":"com.xxx.yyy.MyBoolService",
  "android.intent.action.BOOT_COMPLETED"
  }
  ```

  La representació de cadascuna d'aquestes parelles serà d'un node del tipus que sigui la classe, connectat amb un node del número associat al filtre d'intent, com es mostra a la figura 5.3.

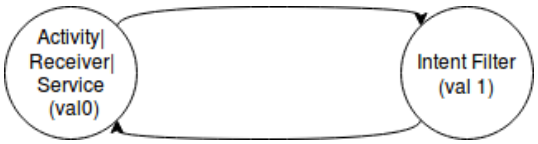


Figura 5.3: Representació gràfica de com es representen els IntentFilters a BadsDroid

- dangerAPIs: Mètodes de l'API d'Android, que a Apposcopy es consideren perilloses. L'estructura és molt semblant a la dels IntentFilters:

  ```
  {
  "val0":"com.xxx.yyy.MyService",
  ```

```
"val1":"javax.crypto.Cipher:byte[] doFinal(byte[])"
}
```

I la seva representació al graf de BadsDroid també és molt similar a la dels IntentFilters (fig. 5.4).

Figura 5.4: Representació gràfica de com es representen les dangerAPIs a BadsDroid

- taintFlows: és el que Ruben Martins destacava en anteriors versions que seria important tenir-ho en compte. En aquest cas no són parelles de classes, sinó que són quatre valors. Dues són classes de l'APK i els altres dos valors la informació que se li demana al sistema operatiu quan es fa la crida entre aquestes classes.

```
{
   "val0":"com.bwlQIUxN.xljNyduA91603.PushService",
   "val1":"$MyTime",
   "val2":"com.bwlQIUxN.xljNyduA91603.OptinActivity",
   "val3":"!INTERNET"
}
```

  La representació en BadsDroid consisteix en dues classes (les dues d'Android) connectades entre elles indirectament, ja que en mig de les seves arestes tenen un node més per aresta, amb cada un dels altres dos valors, com es mostra ala figura 5.5

Amb les representacions esmentades, es substitueixen els nodes pel seu valor numèric i es construeix el graf.

El graf resultant es passa a node2Vec amb el procediment explicat a la segona versió (apartat 5.3.2). Una vegada obtinguts els valors de node2vec,

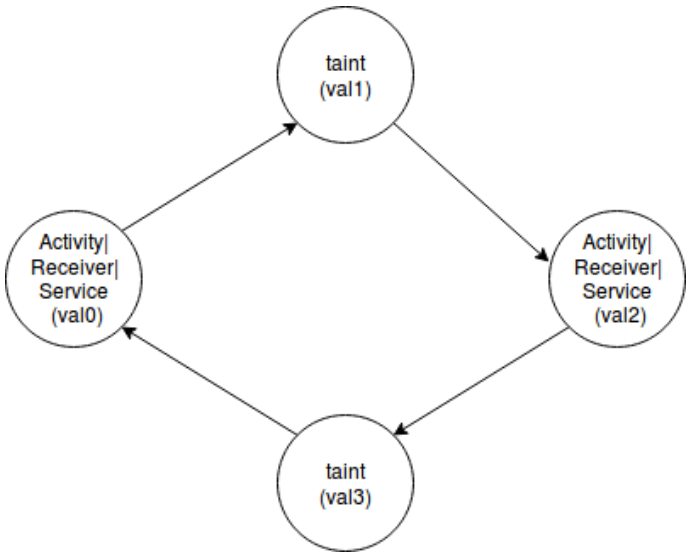


Figura 5.5: Representació gràfica de com es representen els taintFlows a BadsDroid

es fan "random walks" pel graf, d'una llargada determinada, i sense passar més d'una vegada per una mateixa aresta. Es fa un número determinat de camins aleatoris. La ruta de cada camí es representa en forma d'array, i el conjunt d'aquestes array són tots els random walk realitzats. D'aquesta forma obtenim una matriu de dues dimensions. Durant aquest procés es substituïen els valors dels nodes, pels valors obtinguts al node2vec (que són arrays d'una mida determinada). D'aquesta forma el resultat final obtingut són matrius de tres dimensions.

5.3.6 Funcionament final de BadsDroid

En aquesta versió s'implementa la xarxa neuronal amb TensorFlow. La xarxa neuronal consisteix en una capa de normalització, dues capes denses, una dropout [1] i una més densa al final i que funciona per èpoques (1000 en aquest cas).

El funcionament és l'explicat fins ara, consisteix en una eina, que se

[1] Consisteix a desconnectar un percentatge de les neurones a cada iteració de l'entrenament.

li dóna la direcció d'un directori que contingui arxius APK, els examina, prepara les dades i fa entrenar una xarxa neuronal, la qual indica l'accuracy que ha aconseguit amb les dades.

Aquesta xarxa neuronal classifica les APK en tres famílies, DroidKungFu, OpFacke i FakeInstaller.

Després de provar amb múltiples models de xarxes neuronals, finalment amb la que s'han obtingut millors resultats és una amb l'arquitectura següent:

- batch[2]: 30
- shuffle[3]: 500
- cinc capes:
  - capa de batch_normalization
  - dues capes denses
  - capa de dropout
  - última capa, densa

Després de diferents configuracions de la xarxa neuronal, s'ha optat per fer servir aquesta, ja que és amb la que s'han obtingut millors resultats.

[2]En comptes de classificar i actualitzar els pesos per cada mostra, es fa cada X vegades (on X és el valor del batch).

[3]Consisteix a reordenar el dataset aleatòriament cada X vegades, on X és el valor que s'assigna a suffle.

Capítol 6

Resultats

Una vegada finalitzada l'última versió de l'aplicació, podem entrenar la xarxa i validar-la per obtenir la quantitat de vegades que la xarxa neuronal encerta la classe del malware.

El data set que s'ha fet servir és Drebin, explicat més profundament a 6.1. Per fer el training set s'ha fet servir un 80% de les dades obtingudes, i pel set de validació s'ha agafat el 20% restant.

S'han fet proves amb diverses estructures de la xarxa neuronal, però amb la que s'han obtingut milltos resultats és la que s'esmenta a l'apartat anterior.

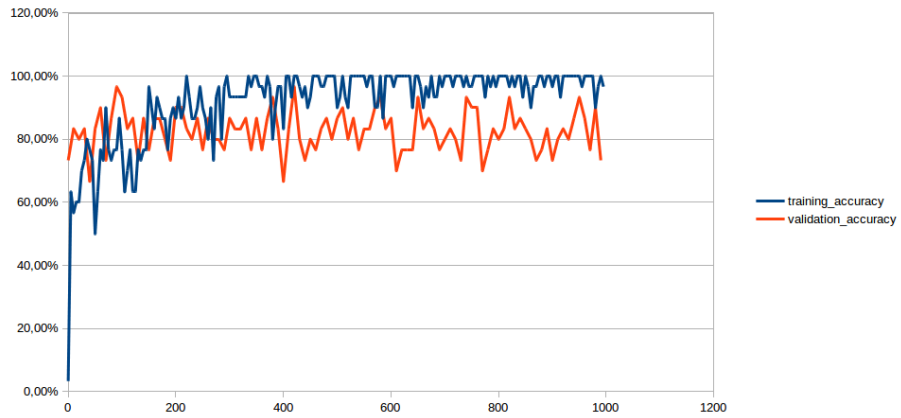


Figura 6.1: Accuracy de l'entrenament i la validació de BadsDroid

Els resultats obtinguts a la validació són d'una mitjana del 82.53%. A la imatge 6.1 es mostra la gràfica de l'accuracy de l'entrenament (blau) per cada 5 èpoques, i es veu com aprèn molt ràpidament. En color taronja es mostren els percentatges d'exactitud de la validació, que oscil·len entre el 65% i el 95% aproximadament.

La pèrdua[1] de la xarxa neuronal va baixant amb els epochs, tal com es pot observar a la figura 6.2.

6.1 Drebin Dataset

Actualment la recerca que es fa per detecció i/o classificació per famílies de malware està molt activa.

Drebin és un mètode lleuger per detectar malware d'Android que permet identificar aplicacions malicioses directament al telèfon intel·ligent. Atès que els recursos limitats impedeixen controlar aplicacions en temps d'execució, DREBIN realitza anàlisis estàtiques, recopilant tantes característiques d'una aplicació com sigui possible. Aquestes característiques estan incrustades en un espai vectorial conjunt, de manera que els patrons típics indicatius per al programari maliciós es poden identificar automàticament i utilitzar-los per prendre les decisions.[41]

Les persones relacionades amb el projecte van decidir crear un repositori semi-públic amb més de 5000 aplicacions malware d'Android per fer servir en l'àrea de la recerca. Aquestes aplicacions estan dividides per famílies de malware Android.

La funció de pèrdua utilitzada consisteix en l'entropia creuada del softmax dels logits i els labels.

[1]La pèrdua és la penalització en la qual s'incorre quan l'estimació de la destinació que proporciona el model de ML no és exactament igual a la destinació.

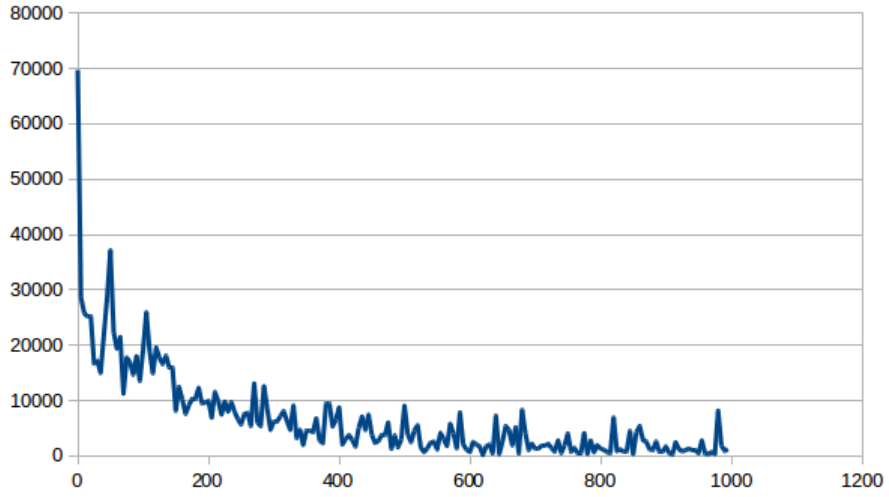


Figura 6.2: Pèrdua durant l'entrenament de la xarxa neuronal de BadsDroid

Capítol 7

Conclusions

Durant la recerca s'ha examinat, implementant i fet servir l'eina BadsDroid per determinar si l'ús de xarxes neuronals per a la classificació de programari maliciós és eficient.

Els resultats obtinguts no són dolents, però hi ha recerques semblants, d'ús de machine learning per a classificació de malware, que han obtingut millors resultats. Aquest fet no implica que aquesta recerca s'hagi de tancar aquí, sinó que cal més temps per poder realitzar més canvis i més proves i continuar amb ella.

Tot i així, veient que en una mitjana del 82.53% dels casos BadsDroid encerta la classificació del malware, es pot concloure que en la majoria de casos, l'ús de xarxes neuronals per a la classificació de malware Android funciona.

Com a conclusió a escala personal, dir que ha sigut molt gratificant fer un treball d'aquestes característiques. M'he sentit molt còmoda i m'ha agradat assolir els reptes als quals m'he hagut d'enfrontar, tot i haver sofert moments d'estrès i de tenir la sensació de no avançar, tot i fer molta feina (que m'agrada fer-ne). Estic bastant satisfeta de la feina feta i dels coneixements adquirits durant el període de realització del treball. Gràcies a la realització d'aquest treball he descobert el món del malware que fins fa poc no m'havia cridat

massa l'atenció i crec que és un món curiós per descobrir i saber que machine learning i malware poden anar agafats de la mà m'ha fet que m'interessi encara més el tema.

Capítol 8

Treball futur

En un futur seria bo inserir noves funcionalitats a BadsDroid, com fer que generi l'aplicació mateixa el graf de flux i de crides, sense dependre d'Apposcopy, ja que Apposcopy fa moltes més accions per darrere i fa servir molts recursos de l'ordinador i de temps.

S'han provat diverses configuracions de la xarxa neuronals, per veure si els resultats milloraven, però no hi ha hagut temps a canviar l'estructura de dades de l'entrada de la xarxa neuronal per provar si milloraven els resultats. Seria convenient fer aquest tipus de proves en un futur.

Estaria molt bé que BadsDroid també es pogués fer servir per detectar si una app és malware o no, (que potser seria tan fàcil com inserir una classificació més). Actualment no s'ha pogut fer ja que no s'han trobat aplicacions legítimes amb les quals Apposcopy funcionés. Quan BadsDroid no depenguis d'Apposcopy aquest pas serà més fàcil.

Seria convenient apropar BadsDroid més als usuaris, i transformar-la a Android o a una aplicació web i que els usuaris puguin saber si un programari és malaware o no.

Bibliografia

[1] Michael Nielsen. Neural networks and deep learning. url http://neuralnetworksanddeeplearning.com/, 2017.

[2] Google. Consumer barometer with google. url https://goo.gl/9CQbyz.

[3] Lauren Guenveur. Ventas de smartphones: Android lidera en un mundo de dos sistemas operativos. url https://goo.gl/3cr5qR.

[4] Juan Carlos Broncano. ¿hay más virus en android que en ios? url https://www.proandroid.com/virus-en-android-y-ios/.

[5] Sophos. Sophoslabs 2018 malware forecast. url https://www.sophos.com/en-us/en-us/medialibrary/PDFs/technical-papers/malware-forecast-2018.pdf?la=en .

[6] Institute for System Security – Technische Universität Braunschweig. Drebin dataset. url https://www.sec.cs.tu-bs.de/ danarp/drebin/index.html.

[7] Yu Feng, Isil Dillig, Saswat Anand, and Alex Aiken. Apposcopy: automated detection of android malware (invited talk). In Aharon Abadi, Rafael Prikladnicki, and Yael Dubinsky, editors, *DeMobile@SIGSOFT FSE*, pages 13–14. ACM, 2014.

[8] Yu Feng, Osbert Bastani, Ruben Martins, Isil Dillig, and Saswat Anand. Automated synthesis of semantic malware signatures using maximum satisfiability. *CoRR*, abs/1608.06254, 2016.

[9] Christian Urcuqui López. Módulo de machine learning para detección de malware en android, 07 2016.

[10] Sebastian Londoño, Christian Urcuqui López, Manuel Fuentes Amaya, Johan Gómez, and Andres Navarro. Safecandy: System for security, analysis and validation in android. 13:89–102, 12 2015.

[11] Asaf Shabtai, Robert Moskovitch, Yuval Elovici, and Chanan Glezer. Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey. *Inf. Secur. Tech. Rep.*, 14(1):16–29, February 2009.

[12] Naser Peiravian and Xingquan Zhu. Machine learning for android malware detection using permission and api calls. In *Proceedings of the 2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, ICTAI '13, pages 300–305, Washington, DC, USA, 2013. IEEE Computer Society.

[13] Zhixing Xu, Sayak Ray, Pramod Subramanyan, and Sharad Malik. Malware detection using machine learning based analysis of virtual memory access patterns. In *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '17, pages 169–174, 3001 Leuven, Belgium, Belgium, 2017. European Design and Automation Association.

[14] Cristina Vatamanu, Doina Cosovan, Dragos Gavrilut, and Henri Luchian. Perceptron-based ensembles and binary decision trees for malware detection. In *Artificial Neural Networks and Machine Learning - ICANN 2017 - 26th International Conference on Artificial Neural Networks, Alghero, Italy, September 11-14, 2017, Proceedings, Part II*, pages 250–259, 2017.

[15] Dragos Gavrilut, Mihai Cimpoesu, Dan Anton, and Liviu Ciortuz. Malware detection using machine learning. In *Proceedings of the International Multiconference on Computer Science and Information Technology*, pages 735–741, 2009.

[16] Blake Anderson, Daniel Quist, Joshua Neil, Curtis Storlie, and Terran Lane. Graph-based malware detection using dynamic analysis. *Journal in Computer Virology*, 7(4):247–258, Nov 2011.

[17] Liu Liu, Bao-sheng Wang, Bo Yu, and Qiu-xi Zhong. Automatic malware classification and new malware detection using machine learning. *Frontiers of Information Technology & Electronic Engineering*, 18(9):1336–1347, Sep 2017.

[18] Nikola Milosevic, Ali Dehghantanha, and Kim-Kwang Raymond Choo. Machine learning aided android malware classification. 61, 02 2017.

[19] N. Peiravian and X. Zhu. Machine learning for android malware detection using permission and api calls. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence(ICTAI)*, volume 00, pages 300–305, Nov. 2013.

[20] Jonh Barker. Malware detection in executables using neural networks.

[21] Joshua Saxe and Konstantin Berlin. Deep neural network based malware detection using two dimensional binary program features. In *Proceedings of the 2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, MALWARE '15, pages 11–20, Washington, DC, USA, 2015. IEEE Computer Society.

[22] Android. Developers guide. url https://developer.android.com/.

[23] ¿qué es el android manifest? url http://www.tuprogramacion.com/glosario/que-es-el-android-manifest/.

[24] Danny Palmer. What is malware? everything you need to know about viruses, trojans and malicious software. url https://goo.gl/4ZTKx7.

[25] Tim Fisher. Malware: What it means, common types, and how to deal with it. url https://www.lifewire.com/what-is-malware-2625933.

[26] programa maligno, mejor que malware. url https://www.fundeu.es/recomendacion/programa-maligno-mejor-malware/.

[27] Michael Sikorski and Andrew Honing. *Practical Malware Analisys*. no starch press, San Francisco, eight printing edition, 2012.

[28] Que es malware. url https://iiemd.com/malware/que-es-malware.

[29] David Pérez. Cómo eliminar el 'virus de la policía' si tu móvil ha sido infectado. url https://elandroidelibre.elespanol.com/2016/01/como-eliminar-virus-de-la-policia.html.

[30] Elena Root i Bogdan Melnykov. Malware displaying porn ads discovered in game apps on google play.

[31] Francisco Salido. Loapi: el malware para android que puede freír tu batería. url https://unaaldia.hispasec.com/2017/12/loapi-el-malware-para-android-que-puede.html.

[32] Redacció BBC. Loapi, el virus de celular que ataca a américa latina y puede arruinar tu teléfono (y cómo evitarlo). url http://www.bbc.com/mundo/noticias-42493219.

[33] Symantec. Android.opfake in-depth. url https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/android-opfake-in-depth-12-en.pdf.

[34] Fernando Ruiz. 'fakeinstaller' leads the attack on android phones. url https://securingtomorrow.mcafee.com/mcafee-labs/fakeinstaller-leads-the-attack-on-android-phones/, 2012.

[35] Kateryna Chumachenko. Machine learning methods for malware detection and classification. url https://goo.gl/5S4o4i.

[36] Vicenç Torra. La inteligencia artificial. url https://goo.gl/nm2Uc9.

[37] Txema Rodriguez. Machine learning y deep learning: cómo entender las claves del presente y futuro de la inteligencia artificial. url https://goo.gl/XYFjCA.

[38] Chris A Mack. *Hands-On Machine Learning with Scikit-Learn & TensorFlow*. O'REILLY, 2017.

[39] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey

Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[40] skylot. jadx. url https://github.com/skylot/jadx.

[41] Daniel Arp, Michael Spreitzenbarth, Hugo Gascon, and Konrad Rieck. Drebin: Effective and explainable detection of android malware in your pocket, 2014.

[42] Hugo Gascon, Fabian Yamaguchi, Daniel Arp, and Konrad Rieck. Structural detection of android malware using embedded call graphs. In *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security*, AISec '13, pages 45–54, New York, NY, USA, 2013. ACM.

[43] Antoine Jean-Pierre Tixier, Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. Classifying graphs as images with convolutional neural networks. *CoRR*, abs/1708.02218, 2017.

[44] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *CoRR*, abs/1709.05584, 2017.

[45] Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels, 2016.

[46] Thomas Kipf. Graph convolutional networks. url https://goo.gl/7pDCGS.

[47] Thang D. Bui, Sujith Ravi, and Vivek Ramavajjala. Neural graph machines: Learning neural networks using graphs. *CoRR*, abs/1703.04818, 2017.

[48] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. *CoRR*, abs/1605.05273, 2016.

[49] Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. Adaptive graph convolutional neural networks. *CoRR*, abs/1801.03226, 2018.

[50] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *CoRR*, abs/1606.09375, 2016.

[51] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *SIGPLAN Not.*, 49(6):259–269, June 2014.

[52] Patrick Lam, Eric Bodden, Laurie Hendren, and Technische Universität Darmstadt. The soot framework for java program analysis: a retrospective.

[53] Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 855–864, New York, NY, USA, 2016. ACM.

[54] TensorFlow. Tensorflow tutorial. url https://www.tensorflow.org/tutorials/.

[55] Jason Brownlee. Develop your first neural network in python with keras step-by-step. url https://goo.gl/v8v5z9.

[56] Antoine Jean-Pierre Tixier, Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. Classifying graphs as images with convolutional neural networks. *arXiv preprint arXiv:1708.02218*, 2017.

[57] L. Li, A. Bartel, T. F. Bissyandé, J. Klein, Y. Le Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Octeau, and P. McDaniel. Iccta: Detecting inter-component privacy leaks in android apps. In *2015 IEEE/ACM*

37th IEEE International Conference on Software Engineering, volume 1, pages 280–291, May 2015.

[58] Elior Cohen. node2vec: Embeddings for graph data. url https://goo.gl/gpnYF9.

[59] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 393–407, Berkeley, CA, USA, 2010. USENIX Association.

[60] Trojan:android/droidkungfu.c. url https://www.f-secure.com/v-descs/trojan_android_droidkungfu_c.shtml.

[61] Andy. Tensorflow dataset api tutorial – build high performance data pipelines. url http://adventuresinmachinelearning.com/tensorflow-dataset-tutorial/.